Snake Game Design & Documentation

Problem Statement

Develop a classic Snake game using Python's Turtle graphics, enabling user interaction through keyboard controls. The game should feature a snake that grows as it eats food items, tracks player scores, and resets upon collision with boundaries or itself. The design should ensure a smooth user experience with responsive controls and clear visual feedback.

Objectives

- Implement a fully functional Snake game with smooth, zigzag movement and multiple food items.
- Provide visual enhancements such as gradient background, snake eyes, and flickering tongue.
- Maintain and display current score and high score.
- Ensure game resets properly after collisions.
- Structure code for modularity, readability, and potential testing.

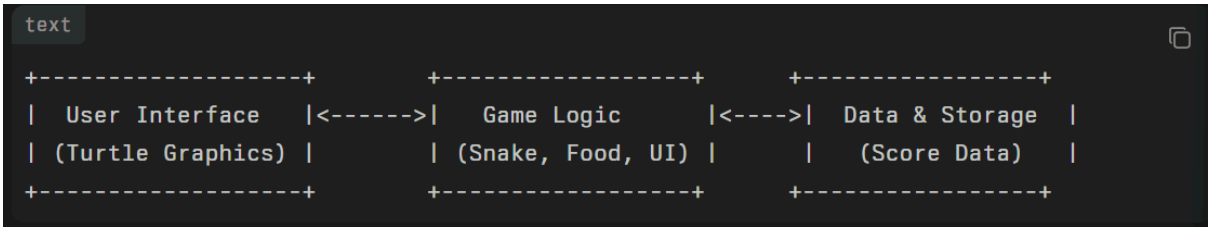
Functional Requirements

- The snake moves in four directions controlled by keys W (up), S (down), A (left), and D (right).
- Snake grows longer by eating any of three food items randomly placed on the screen.
- The snake's movement includes a zigzag offset mechanic for visual effect.
- Score increases by 10 points per food consumed; high score updates accordingly.
- The game resets if the snake hits the screen boundary or its own body.
- Visually update the snake's eyes and tongue direction based on movement.
- Display score and high score prominently on screen.

Non-functional Requirements

- The game should maintain a consistent frame rate with controlled delay.
- Visual elements must be clear and aesthetically pleasing with appropriate colors.
- Controls should be responsive to user input.
- The code should be maintainable and modular to allow future extensions.
- The game should handle missing image assets gracefully.

System Architecture Diagram



- User Interface: Handles all drawing and user input.
- Game Logic: Contains movement, collision detection, scoring, and reset logic.
- Data & Storage: Tracks scores and game state.

Process Flow or Workflow Diagram

1. Initialize game screen and draw background.
2. Create snake, food sprites, and UI elements.
3. Bind keyboard controls for movement.
4. Main game loop:
    - Update snake position with zigzag.
    - Check food collisions:
        - Reposition food
        - Grow snake
        - Update score and delay
    - Check boundary and self collision:
        - Reset game if collision detected
    - Update visuals (eyes, tongue).
    - Refresh display and wait for next frame.

UML Diagrams

Use Case Diagram

- Actors: Player
- Use Cases: Move Snake, Eat Food, Grow Snake, Update Score, Reset Game, View Scores

Class Diagram (Simplified)

Sequence Diagram (Simplified)

Player → Game Loop: Input direction (W/A/S/D)
Game Loop → Snake: move()
Game Loop → Food: check_collision()
Game Loop → Snake: grow() if collision
Game Loop → ScoreBoard: update()
Game Loop → Snake: check_collision() (self/border)
Game Loop → Snake: reset() if collision

Database/Storage Design

Not applicable as the game does not use persistent storage beyond runtime variables. Scores are stored in memory and lost on program exit.
